Übung 1: Entwurfsmethoden und Analyse von Algorithmen

Aufgabe 1-1:

Schreiben Sie ein Programm, dass den Binomialkoeffizienten $\binom{n}{k}$ einmal rekursiv und einmal iterativ berechnet. Zur Erinnerung:

$$\binom{n}{k} = \begin{cases} 1 & \text{falls } n = k \text{ oder } k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sonst} \end{cases}$$

Welche Laufzeiten haben die einzelnen Berechnungen? Sind die Laufzeiten polynomiell?

Aufgabe 1-2:

In dieser Aufgabe soll untersucht werden, welche Auswirkung die Modellierung bzw. Repräsentation auf die Laufzeit hat.

Ein Polynom $p(x) = c_n \cdot x^n + c_{n-1} \cdot x^{n-1} + \ldots + c_1 \cdot x^1 + c_0$ vom Grad n kann mittels der Koeffizienten $c_0, \ldots, c_n$ dargestellt werden. Wenn das Polynom n reelle Nullstellen hat, dann kann das Polynom aber auch mittels Linearfaktoren als $p(x) = r_0 \cdot (x - r_1) \cdot (x - r_2) \ldots (x - r_n)$ dargestellt werden, wobei $r_1, \ldots, r_n$ die Nullstellen sind. In Abbildung 1 ist das Polynom $p(x) = 2 \cdot (x + 1)(x - 1)(x - 3) = 2x^3 - 6x^2 - 2x + 6$ mit den Nullstellen -1, 1 und 3 dargestellt.

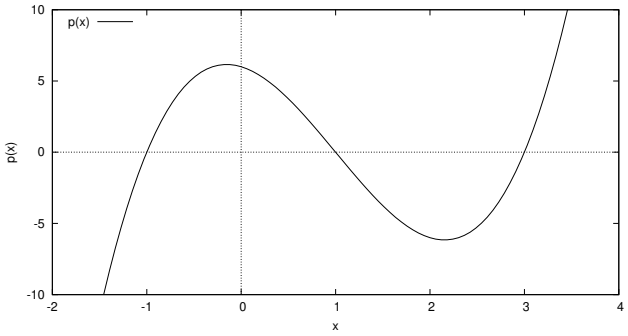

Abbildung 1: Polynom vom Grad 3 mit 3 reellen Nullstellen

Zwei Polynome $p = \sum_{i=0}^{n} a_i x^i$ und $q = \sum_{i=0}^{m} b_i x^i$ werden wie folgt multipliziert:

$$(p \cdot q)(x) = p(x) \cdot q(x) = \sum_{i=0}^{n} a_i x^i \cdot \sum_{i=0}^{m} b_i x^i = \sum_{k=0}^{n+m} \Big(\sum_{i+j=k} a_i b_j \Big) x^k$$

Schreiben Sie ein Programm, mit dem Polynome erzeugt und multipliziert werden können sowie der Wert eines Polynoms an einer Stelle x bestimmt werden kann. Welche Laufzeit hat die Auswertung an einer Stelle x sowie die Multiplikation zweier Polynome mit den jeweiligen Repräsentationen?

Aufgabe 1-3:

Implementieren Sie jeweils einen rekursiven Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier ganzer Zahlen a und b:

$$\gcd(a, b) = \begin{cases} \gcd(a, b - a) & \text{falls } a < b \\ \gcd(a - b, b) & \text{sonst} \end{cases} \qquad \text{oder} \qquad \gcd(a, b) = \gcd(b, a \bmod b)$$

Ist eine der beiden Zahlen a oder b gleich null, dann ist das Rekursionsende erreicht und das Ergebnis ist der Wert der anderen Zahl.

Welche Laufzeit haben die beiden Algorithmen, wenn wir die Anzahl der mathematischen Operationen zählen? Welche Laufzeit haben die Programme?

Aufgabe 1-4:

Zeigen oder widerlegen Sie folgende Aussagen:

1. $2 \cdot n^2 + 37 \cdot n^3 \in \mathcal{O}(n^3)$
2. $2 \cdot n^2 + 37 \cdot n^3 \in \mathcal{O}(n^2)$
3. $2 \cdot n^2 + 37 \cdot n^3 \in \mathcal{O}(n^4)$
4. $2 \cdot n^2 + 37 \cdot n^3 \in \Omega(n^3)$
5. $2 \cdot n^2 + 37 \cdot n^3 \in \Omega(n^2)$
6. $2 \cdot n^2 + 37 \cdot n^3 \in \Omega(n^4)$
7. $\frac{2n^2}{n+1} \in \mathcal{O}(n)$
8. $17 \cdot \sqrt{n} + 139 \cdot n \in \Theta(n)$
9. $17 \cdot \sqrt{n} + 139 \cdot n \in \Theta(\sqrt{n})$
10. $c_0 + c_1 \cdot n + c_2 \cdot n^2 + \ldots + c_k n^k \in \mathcal{O}(n^k)$, mit $c_i \in \mathbb{R}, i = 1, \ldots, k-1$ und $c_k \in \mathbb{R}_+$.
11. $\mathcal{O}(n^{\frac{5}{2}}) = \mathcal{O}\left(\frac{n^5}{n^2}\right)$
12. $\mathcal{O}(\log_2(n)) = \mathcal{O}(\log_{10}(n))$

Aufgabe 1-5:

Widerlegen Sie durch Gegenbeispiele die folgenden Aussagen:

1. Für jede Funktion $f : \mathbb{N} \to \mathbb{R}^+$ gilt entweder $f \in \mathcal{O}(n^2)$ oder $f \in \Omega(n^2)$ oder beides.
2. Ein Algorithmus mit Laufzeit in $\Theta(n^2)$ ist für alle Eingabegrößen schneller als ein Algorithmus mit Laufzeit in $\Theta(3^n)$.

Aufgabe 1-6:

Zeigen Sie folgende Aussagen:

1. $\mathcal{O}(n^k) \subsetneq \mathcal{O}(e^n)$ für ein konstantes $k \in \mathbb{N}$
2. $\mathcal{O}(\ln^k(n)) \subsetneq \mathcal{O}(n)$ für ein konstantes $k \in \mathbb{N}$

D.h. ein Polynom mit beliebigem Exponenten steigt langsamer als eine Exponentialfunktion und ein Logarithmus mit beliebigem Exponenten steigt langsamer als eine lineare Funktion.

Hinweise: $\mathcal{O}(f) \subsetneq \mathcal{O}(g)$ bedeutet $O(f)$ ist eine echte Teilmenge von $\mathcal{O}(g)$ oder anders ausgedrückt, $f \in \mathcal{O}(g)$ und $g \notin \mathcal{O}(f)$.

Regel von l'Hospital:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

Es gelten folgende Ableitungsregeln:

$$\begin{aligned}
\left(n^k\right)' &= k \cdot n^{k-1} \\
(e^n)' &= e^n \\
(\ln(n))' &= \frac{1}{n} \\
\left(\ln^k(n)\right)' &= \left((\ln(n))^k\right)' = k \cdot \left((\ln(n))^{k-1}\right) \cdot \frac{1}{n}, \quad \text{(Kettenregel)}
\end{aligned}$$

Aufgabe 1-7:

In dieser Aufgabe soll ein Algorithmus entwickelt werden, der die Dominanzzahl von Punkten berechnet. Gegeben sind n Punkte $M = \{p_1, \ldots, p_n\}$ in einem $N \times N$-Gitter, wobei jeder Punkt $p = (p.x, p.y)$ durch seine x- und y-Koordinaten gegeben ist.

Die Dominanzzahl $\mathrm{dz}(p, M) = |\{q \in M \mid q.x < p.x \text{ und } q.y \leq p.y\}|$ eines Punktes $p \in M$ ist die Anzahl der Punkte aus M, die „links unterhalb“ des Punktes p liegen.

Geben Sie einen möglichst effizienten Algorithmus an, der die Dominanzzahlen aller Punkte berechnet.

Aufgabe 1-8:

Schreiben Sie ein Programm zur Berechnung arithmetischer Ausdrücke. Wie solche Ausdrücke mittels Syntaxdiagrammen dargestellt werden, ist in Abbildung 2 zu sehen.

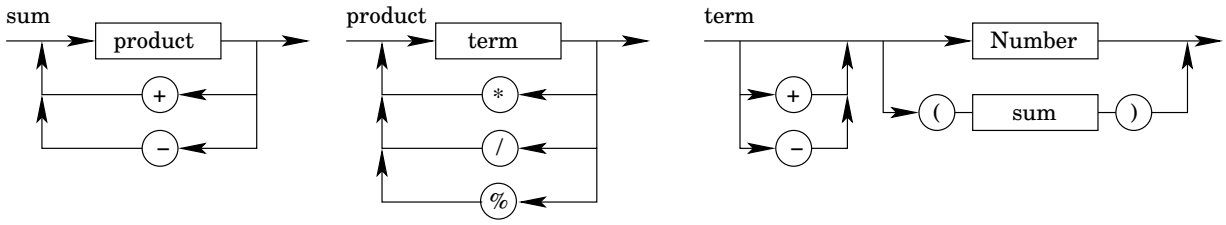

Abbildung 2: Syntaxdiagramm für arithmetische Ausdrücke

Eine Summe setzt sich aus Produkten zusammen, die jeweils mit plus oder minus verknüpft werden. Ein Produkt setzt sich aus Termen zusammen, die jeweils durch Multiplikation oder Division verknüpft werden. Ein Term ist eine Zahl oder ein geklammerter Ausdruck. Wollen wir eine Auswertung implementieren, die diesem Schema folgt, realisieren wir `sum`, `product` und `term` durch Funktionen, die sich gegenseitig aufrufen.

Aufgabe 1-9:

In der Vorlesung wurde der Algorithmus DynamicKP zur Lösung des 0/1-Rucksackproblems mittels dynamischer Programmierung vorgestellt. Erweitern Sie den Algorithmus so, dass auch die ausgewählten Objekte ausgegeben werden.

Aufgabe 1-10:

Im folgenden Algorithmus sei DoSomething eine Funktion mit konstanter Laufzeit. Leiten Sie die asymptotische Laufzeit (Groß-Theta) der Funktion Foo her.

```
1: function Foo(n : Integer)
2:     for i := 1, ..., n do
3:         j := 1
4:         while j · j < n do
5:             DoSomething(i, j)
6:             j := j + 1
```

Aufgabe 1-11:

In den folgenden drei Algorithmen liefert die Funktion **max** in konstanter Zeit den maximalen Wert der zwei Parameter; das Array a steht global zur Verfügung. Leiten Sie die asymptotische Laufzeit (Groß-Theta) der Funktionen Max1, Max2 und Max3 her.

function Max1(ℓ, r : Integer) : Integer

```
    if ℓ = r then
        return a[ℓ]
    if r − ℓ = 1 then
        return max(a[ℓ], a[r])
    m := ⌊(ℓ + r)/2⌋
    return max( Max1(ℓ, m), Max1(m + 1, r))

function Max2(n : Integer) : Integer
    for i := 1, . . . , n − 1 do
        for j := i + 1, . . . , n do
            if a[i] > a[j] then
                t := a[i], a[i] := a[j], a[j] := t
    return a[n]

function Max3(n : Integer) : Integer
    res := a[1]
    for i := 2, . . . , n do
        res := max(a[i], res)
    return res
```

Aufgabe 1-12:

Im folgenden Algorithmus sei a ein globales Array der Länge n, das ganze Zahlen (Integer) speichert. Was liefert der Aufruf $\textsc{g}(1, n)$? Schätzen Sie außerdem die Anzahl der Additionen (Zeile 5) in Abhängigkeit von ℓ und r ab.

```
1: function g(ℓ, r : Integer) : Integer
2:     if ℓ > r then
3:         return 0
4:     m := ⌊(ℓ+r)/2⌋
5:     return g(ℓ, m − 1) + a[m] + g(m + 1, r)
```

Aufgabe 1-13:

Welche asymptotische Laufzeit (Groß-Theta) hat die folgende Funktion und geben Sie für $n = 1, 2, 4, 8$ jeweils die Werte $\textsc{f}(n)$ an.

```
function f(n : Integer) : Integer
    r := 0
    for i := 1, . . . , n do
        r := r + 1
        j := 1
        while j ≤ n do
            r := r + 1
            j := j · 2
    return r
```